

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7      "math"
8  )
9
10 type (
11     lmnt struct {
12         val int
13         pos int
14     }
15     deque []lmnt
16 )
17
18 func initDeque() deque {
19     return make([]lmnt, 0, 0)
20 }
21
22 func (dq *deque) PushLeft (w lmnt) {
23     *dq = append([]lmnt{w}, (*dq)...)
24 }
25
26 func (dq *deque) PushRight (w lmnt) {
27     *dq = append(*dq, w)
28 }
29
30 func (dq deque) GetLeft() lmnt {
31     if dq.Empty() {
32         return lmnt{math.MinInt64, -1}
33     } else {
34         return dq[0]
35     }
36 }
37
38 func (dq deque) GetRight() lmnt {
39     if dq.Empty() {
40         return lmnt{math.MinInt64, -1}
41     } else {
42         return dq[len(dq)-1]
43     }
44 }
45
46 func (dq *deque) RemoveLeft () error {
47     if (*dq).Empty() {
48         return errors.New("Attempt to remove from empty deque")
49     } else {
50         *dq = (*dq)[1:]
51         return nil
52     }
53 }
54
55 func (dq *deque) RemoveRight () error {
56     if (*dq).Empty() {
57         return errors.New("Attempt to remove from empty deque")
58     } else {
59         *dq = (*dq)[:len(*dq)-1]
60         return nil
61     }
62 }
63
64 func (dq deque) Empty() bool {
65     return len(dq) == 0
66 }

```

```

67
68 func main() {
69     f, err := os.Open("numbers.dat")
70     if err != nil {
71         return
72     }
73     defer f.Close()
74     var k int
75     _, err = fmt.Fscanf(f, "%d\n", &k)
76     if err != nil { return }
77     var x int
78     win := initDeque()
79     for i := 0; i < k; i++ {
80         _, err := fmt.Fscanf(f, "%d\n", &x)
81         if err != nil { break }
82
83         for !win.Empty() && win.GetRight().val <= x {
84             win.RemoveRight()
85         }
86         win.PushRight(lmnt{x, i})
87     }
88     fmt.Println(win.GetLeft().val)
89
90     for i := k; ; i++ {
91         _, err := fmt.Fscanf(f, "%d\n", &x)
92         if err != nil { break }
93
94         if win.GetLeft().pos == i-k {
95             win.RemoveLeft()
96         }
97
98         for !win.Empty() && win.GetRight().val <= x {
99             win.RemoveRight()
100         }
101         win.PushRight(lmnt{x, i})
102
103         fmt.Println(win.GetLeft().val)
104     }
105 }

```

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7      "math"
8  )
9
10 type (
11     data struct {
12         val int
13         pos int
14     }
15     lmnt struct {
16         d data
17         l *lmnt
18         r *lmnt
19     }
20     deque struct {
21         left *lmnt
22         right *lmnt
23     }
24 )
25
26 func initDeque() deque {
27     return deque{nil, nil}
28 }
29
30 func (dq *deque) PushLeft (d data) {
31     if (*dq).Empty() {
32         (*dq).left = &lmnt{d, nil, nil}
33         (*dq).right = (*dq).left
34     } else {
35         ((*dq).left).l = &lmnt{d, nil, (*dq).left}
36         (*dq).left = ((*dq).left).l
37     }
38 }
39
40 func (dq *deque) PushRight (d data) {
41     if (*dq).Empty() {
42         (*dq).left = &lmnt{d, nil, nil}
43         (*dq).right = (*dq).left
44     } else {
45         ((*dq).right).r = &lmnt{d, (*dq).right, nil}
46         (*dq).right = ((*dq).right).r
47     }
48 }
49
50 func (dq deque) GetLeft() data {
51     if dq.Empty() {
52         return data{math.MinInt64, -1}
53     } else {
54         return (*dq.left).d
55     }
56 }
57
58 func (dq deque) GetRight() data {
59     if dq.Empty() {
60         return data{math.MinInt64, -1}
61     } else {
62         return (*dq.right).d
63     }
64 }
65

```

```

66 func (dq *deque) RemoveLeft () error {
67     if (*dq).Empty() {
68         return errors.New("Attempt to remove from empty deque")
69     }
70     if (*dq).left == (*dq).right {
71         (*dq).left, (*dq).right = nil, nil
72     } else {
73         (*dq).left = ((*dq).left).r
74     }
75     return nil
76 }
77
78 func (dq *deque) RemoveRight () error {
79     if (*dq).Empty() {
80         return errors.New("Attempt to remove from empty deque")
81     }
82     if (*dq).left == (*dq).right {
83         (*dq).left, (*dq).right = nil, nil
84     } else {
85         (*dq).right = ((*dq).right).l
86     }
87     return nil
88 }
89
90 func (dq deque) Empty() bool {
91     return dq.left==nil
92 }
93
94 func main() {
95     f, err := os.Open("numbers.dat")
96     if err != nil { return }
97     defer f.Close()
98     var k int
99     _, err = fmt.Fscanf(f, "%d\n", &k)
100    if err != nil { return }
101    var x int
102    win:= initDeque()
103    for i:= 0; i < k; i++ {
104        _, err := fmt.Fscanf(f, "%d\n", &x)
105        if err !=nil { break }
106
107        for !win.Empty() && win.GetRight().val <= x {
108            win.RemoveRight()
109        }
110        win.PushRight(data{x, i})
111    }
112    fmt.Println(win.GetLeft().val)
113
114    for i:= k; ; i++ {
115        _, err := fmt.Fscanf(f, "%d\n", &x)
116        if err != nil { break }
117
118        if win.GetLeft().pos == i-k {
119            win.RemoveLeft()
120        }
121
122        for !win.Empty() && win.GetRight().val <= x {
123            win.RemoveRight()
124        }
125        win.PushRight(data{x, i})
126        fmt.Println(win.GetLeft().val)
127    }
128 }

```

numbers_04.go

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "math"
7  )
8
9  type (
10     lmnt int
11     ring struct {
12         row []lmnt
13         currentPos int
14     }
15 )
16
17 func initRing(len int) ring {
18     var r ring
19     r.row = make([]lmnt, len, len)
20     r.currentPos = 0
21     return r
22 }
23
24 func (r ring) Empty() bool {
25     return r.Length() == 0
26 }
27
28 func (r ring) Length() int {
29     return len(r.row)
30 }
31
32 func (r ring) GetCurrent() lmnt {
33     if r.currentPos >= r.Length() || r.currentPos < 0 {
34         return math.MinInt64
35     } else {
36         return r.row[r.currentPos]
37     }
38 }
39
40 func (r *ring) RemoveCurrent () error {
41     if r.currentPos >= r.Length() || r.currentPos < 0 {
42         return errors.New("Invalid removing")
43     } else {
44         (*r).row = Delete((*r).row, (*r).currentPos)
45         return nil
46     }
47 }
48
49 func Delete(s []lmnt, i int) []lmnt {
50     return s[:i+copy(s[i:], s[i+1:])]
51 }
52
53 func (r *ring) Forward (step int) error {
54     if (*r).Empty() {
55         return errors.New("Invalid step")
56     } else {
57         (*r).currentPos = ((*r).currentPos + step) % (*r).Length()
58         return nil
59     }
60 }
61

```

```

62 func main() {
63     var N, M int
64     fmt.Print("Enter N: ")
65     fmt.Scanf("%d\n", &N)
66     fmt.Print("Enter M: ")
67     fmt.Scanf("%d\n", &M)
68
69     r := initRing(N)
70     for i, _ := range r.row {
71         r.row[i] = 1mnt(i+1)
72     }
73
74     for i := 0; i < N; i++ {
75         r.Forward(M-1)
76         fmt.Println(r.GetCurrent())
77         r.RemoveCurrent()
78     }
79 }

```

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "math"
7  )
8
9  type (
10     lmnt struct {
11         n int
12         next *lmnt
13     }
14     ring struct {
15         current *lmnt
16         len int
17     }
18 )
19
20 func initRing(len int) ring {
21     return ring {nil, 0}
22 }
23
24 func (r ring) Empty() bool {
25     return r.Length() == 0
26 }
27
28 func (r ring) Length() int {
29     return r.len
30 }
31
32 func (r ring) GetCurrentValue() int {
33     if r.current == nil {
34         return math.MinInt64
35     } else {
36         return (*r.current).n
37     }
38 }
39
40 func (r *ring) RemoveNext () error {
41     if r.Empty() {
42         return errors.New("Invalid removing")
43     }
44     if r.Length() > 1 {
45         ((*r).current).next = ((*r).current).next.next
46     } else {
47         // r.Length() == 1
48         (*r).current = nil
49     }
50     (*r).len--
51     return nil
52 }
53
54 func (r *ring) PopNext () (int, error) {
55     if r.Empty() {
56         return 0, errors.New("Invalid removing")
57     }
58     val:= ((*r).current).next.n
59     return val, (*r).RemoveNext()
60 }
61

```

```

62 func (r *ring) InsertFront (value int) {
63     if r.Empty() {
64         (*r).current = new(lmnt)
65         (*(*r).current).n = value
66         (*(*r).current).next = (*r).current
67     } else {
68         p:= &lmnt{value, (*(*r).current).next}
69         (*(*r).current).next = p
70     }
71     (*r).len++
72 }
73
74 func (r *ring) OneStep () error {
75     if (*r).Empty() {
76         return errors.New("Invalid step")
77     } else {
78         (*r).current = (*(*r).current).next
79         return nil
80     }
81 }
82
83 func main() {
84     var N, M int
85     fmt.Print("Enter N: ")
86     fmt.Scanf("%d\n", &N)
87     fmt.Print("Enter M: ")
88     fmt.Scanf("%d\n", &M)
89
90     r:= initRing(N)
91     for i:= N; i > 0; i-- {
92         r.InsertFront(i)
93     }
94
95     for i:= 0; i < N; i++ {
96         for j:= 0; j < (M -1)% r.Length(); j++ { r.OneStep() }
97         if tmp, err:= r.PopNext(); err == nil {fmt.Println(tmp)}
98     }
99 }

```

josephus_02.go